ELSEVIER

Theoretical Computer Science 231 (2000) 5–15

Theoretical Computer Science

www.elsevier.com/locate/tcs

LANGAGE: A maple package for automaton characterization of regular languages

Pascal Caron

Laboratoire d'Informatique Fondamentale et Appliquée de Rouen, Université de Rouen, 76821 Mont-Saint Aignan, Cedex, France

Abstract

LANGAGE is a set of procedures for deciding whether or not a language given by its minimal automaton is piecewise testable, locally testable, strictly locally testable, or strongly locally testable. New polynomial algorithms are implemented for the two last properties. This package is written using the symbolic computation system Maple. It works with AG, a set of Maple packages for processing automata and finite semigroups. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Algorithms; Automata; Locally testable languages; Regular languages

1. Introduction

Automata and regular languages theory is at the heart of theoretical computer science. In the past 30 years a lot of research work has been devoted to classifying regular languages. Schützenberger has been a precursor in this domain, especially with his results on star-free events [9]. Several subclasses of star-free languages have been characterized through properties of their semigroup. It is the case for piecewise testable languages (PT) studied by Simon [10], locally testable languages (LT) investigated by both McNaughton [7] and by Brzozowski and Simon [3], and for strongly locally testable languages (SLT) introduced by Beauquier and Pin [2]. From an algorithmic point of view, these algebraic characterizations lead to procedures with a high time complexity, since computing the syntaxic semigroup of a language is exponential on the number of states of its minimal automaton.

The tests implemented in the LANGAGE package are based on properties of the minimal automaton; time complexity is polynomial. The characterization of piecewise testable automata is due to Simon [10], and the related algorithm to Stern [11]. For

E-mail address: caron@dir.univ-rouen.fr.

PII: S0304-3975(99)00013-4

locally testable automata, we implement the algorithm of Kim et al. [6]. The algorithms dealing with strictly locally testable automata and strongly locally testable automata are deduced from recent characterizations given by the author in [4]. This new Maple package is a contribution to the development of the programming project Automate.[1] The aim of this project is to provide a symbolic computation system on automata, languages, semigroups and words. This paper contains four sections including this one. Section 2 presents the needful theory to understand algorithms. Section 3 is dedicated to algorithms. The last section allows us to conclude.

Most proofs in this paper are just sketches. The complete proof can be found in the full version [4].

2. Theoretical background

In this section, for each family of languages one can study using the LANGAGE package procedures, we provide a formal definition, as well as a characterization based on automata properties.

Let us first recall that a finite state automaton $\mathscr{M}$ is a 5-tuple $(\Sigma, Q, i, F, \delta)$ where Σ is a finite alphabet, Q is a finite set of states, $i \in Q$ is the initial state, $F \subseteq Q$ is the set of terminal states, $\delta : Q \times \Sigma \to Q$ is the transition function. We shall use the term *connected component* (CC) to refer to any subgraph whose underlying undirected graph is connected. By SCC we mean a strongly connected component. An SCC C_1 is an ancestor (resp. descendant) of an SCC C_2 if there exists a path from C_1 to C_2 (resp. from C_2 to C_1). We will use a classical algorithm described in [1] to find all SCCs of a state transition graph.

Three procedures of the LANGAGE package are devoted to the study of local testability. This notion is classically illustrated [2] by considering a window of small size, which is moved along the input word, so that information may be logged from strings appearing in the window, without care of their number nor of their order. Different kinds of local testability can be described by means of variants of this mechanism.

2.1. *Strictly locally testable languages* (*sLT*)

For strictly local testability, a window of size k scans the input word in order to verify that its prefix of length k is a good one, all interior factors of length k are good ones and its suffix of length k is a good one. More precisely, we will use the definition given by McNaughton and Papert in [8].

Definition 2.1. Let k be a positive integer. For $w \in \Sigma^+$ of length $\geqslant k$, let $L_k(w)$, $R_k(w)$ and $I_k(w)$ be respectively the prefix of length k, the suffix of length k and the set

[1] Automate software development project is carried on by A.I.A. (Algorithmics and Implementation of Automata) working group, L.I.R. laboratory (contact: {Jean-Marc.Champarnaud, Djelloul.Ziadi}@dir.univ-rouen.fr).

of interior factors of length k of the word w. $L \subseteq \Sigma^*$ is strictly k-testable if and only if there exist three sets X, Y, Z of words on Σ such that for all $w \in \Sigma^+$, $|w| \geqslant k$, $w \in L$ iff $L_k(w) \in X$, $R_k(w) \in Y$ and $I_k(w) \subseteq Z$. A language is strictly locally testable if it is strictly k-testable for some $k > 0$.

Definition 2.2 (*Local automaton*). Let $\mathscr{M} = (\Sigma, Q, i, F, \delta)$ be a deterministic automaton and let k be a positive integer. $\mathscr{M}$ is k-local if, $\forall w \in \Sigma^k$, the set $\{q.w \mid q \in Q\}$ contains at most one element. An automaton is local if it is k-local for some $k > 0$.

Theorem 2.3. *A language is strictly locally testable iff its trim minimal automaton is local.*

Proof. For the only if part of the statement, we will provide a reductio ad absurdum. Consider a strictly locally testable language L and a word w (long enough) leading to two different states (say q and q') on the minimal automaton of L. The strict locality implies that all successful path which contains w as factor have the same suffixes. By minimality it implies that $q = q'$. The if part is shown by induction on the length of the words of the language. □

Definition 2.4 (*s-local*, *pairwise s-local*). Let $\mathscr{M} = (\Sigma, Q, i, F, \delta)$ be an automaton.

1. A strongly connected component (SCC) C of the state transition graph of $\mathscr{M}$ is s-local if and only if there do not exist two distinct states p and q in C and a word w in Σ^+ such that $\delta(p, w) = p$ and $\delta(q, w) = q$.
2. Let C_1 and C_2 be two disctinct SCCs; then C_1 and C_2 are pairwise s-local iff there do not exist two distinct states p and q respectively in C_1 and C_2 and a word $w \in \Sigma^+$ such that $\delta(p, w) = p$ and $\delta(q, w) = q$.

The algorithm we have implemented for testing whether a language is strictly locally testable or not is deduced from the following theorem.

Theorem 2.5. *A language is strictly locally testable iff the state transition graph of its trim minimal automaton has the following properties*:

1. *All SCCs of the state transition graph are s-local.*
2. *All SCCs of the state transition graph are pairwise s-local.*

Proof. If there exist a word w and two states p and q such that $\delta(p, w) = p$ and $\delta(q, w) = q$ then for all w long enough, the set $\{q \cdot w \mid w \in Q\}$ has more than one element. For the converse, we consider two distinct paths of length $\geqslant n^2$, where n is the number of states of the minimal automaton with the same label and we show that there exists one pair of states encountered twice. □

2.2. Locally testable languages

A locally testable language L is a language with the property that, for an integer k, whether or not a word u is in the language depends (1) on the prefix and suffix of

the word u of length $k-1$ and (2) on the set of intermediate substrings of length k of the word u. A more formal definition is proposed by Zalcstein in [12].

Definition 2.6 (LT). A k-testable language is a boolean combination of strictly k-testable languages. A language is locally testable if it is k-testable for some $k > 0$.

Kim et al. [6] describe a polynomial algorithm making it possible to decide whether a minimal deterministic automaton recognizes a locally testable language or not. We list here some definitions and theorems on which this algorithm is based.

Definition 2.7 (*Transition span*). Let $\mathscr{M} = (\Sigma, Q, i, F, \delta)$ be an automaton and let C be a connected component of the state transition graph of $\mathscr{M}$. The transition span in C of a state $p \in Q$ is the set $TS(C, p) = \{x, x \in \Sigma^* \mid$ for every prefix w of x, $\delta(p, w)$ is in $C\}$.

Definition 2.8 (*TS-equivalence*). Let C be a CC of the state transition graph of an automaton. States p and q are TS-equivalent in C iff $TS(C, p) = TS(C, q)$.

Definition 2.9 (*TS-local*). Let C_j be an SCC of the state transition graph of a deterministic automaton. The graph is TS-local w.r.t. C_j if and only if it has the following property, for every SCC C_i which is an ancestor of C_j, either

- C_i and C_j are pairwise s-local, or otherwise
- there exists a pair of states p and q respectively in C_i and C_j such that p and q are TS-equivalent in C_{ij} (the reaching component from C_i to C_j).

The state transition graph of an automaton is TS-local if and only if for every SCC C_j of the graph, it is TS-local w.r.t. C_j.

Theorem 2.10 (Characterization). *A minimal deterministic automaton is locally testable if and only if it satisfies the following conditions*:

1. *All SCCs of the state transition graph are s-local.*
2. *The state transition graph is TS-local.*

2.3. Strongly locally testable languages

The notion of strongly locally testable languages is a variation of the notion of locally testable languages. Only substrings of length k of a word u are needed to know whether or not this word belongs to a strongly locally testable language. The following definition is quite usual [2].

Definition 2.11. L is strongly locally testable if it is made up with a finite boolean combination of languages of the form $\Sigma^* w \Sigma^*$ where $w \in \Sigma^*$.

Definition 2.12. Let $\mathscr{M} = (\Sigma, Q, i, F, \delta)$ be an automaton and C be an SCC of the state transition graph of $\mathscr{M}$. Let L_C^k be the language relative to the strongly connected

component C and to the positive integer k:

$$L_C^k = \{u \mid |u| > k \text{ and } \exists x \in C \text{ such that } \delta(x,u) \in C\}.$$

The prefix language of the SCC C denoted by P_C is defined as follows:

$$P_C = \{u \mid \exists v \in \Sigma^* \text{ such that } \delta(i, u \cdot pv) \in C\}.$$

Theorem 2.13. *Let L be a language and $\mathscr{M} = (\Sigma, Q, i, F, \delta)$ be its minimal automaton. L is strongly locally testable iff the following conditions are verified*:
1. *L is locally testable.*
2. *For every SCC C, $\forall p, q \in C$, $p \in F \Leftrightarrow q \in F$.*
3. *For every SCC C of the state transition graph of $\mathscr{M}$, $\exists k$ such that $L_C^k \subseteq P_C$ or $L_C^k \cap P_C = \emptyset$.*

Proof. The proof of Theorem 2.13 is in three steps. First we prove that (3) is equivalent to say that two words x, y having their image in the same $\mathscr{D}$-class implies that these words are labels of two paths leading to the same SCC. Then we show that a strongly locally testable language has this last property. And last we prove by (1), (2) and (3) that the $\mathscr{D}$-classes of the syntaxic semigroup are saturated by the language (i.e. the language is strongly locally testable). See [2]. □

In the following, we will write L_C for L_C^k whenever it is not ambiguous.

2.4. Piecewise testable languages

Definition 2.14. A language is piecewise testable if it is a boolean combination of languages of the form $\Sigma^* a_1 \Sigma^* a_2 \ldots \Sigma^* a_n \Sigma^*$ where $a_i \in \Sigma, i = 1, \ldots, n$.

The minimal automaton of a piecewise testable language has been characterized by Simon [10]. In order to state Simon's result we need some additional definitions.

Let $G = (Q, \delta)$ be the state transition graph of an automaton $\mathscr{M}$, and Γ a subset of Σ. The state transition graph of $\mathscr{M}$ on Γ is the graph $G_\Gamma = (Q, \delta_\Gamma)$ such that $\delta_\Gamma = \{(x, a, y) \in \delta \mid a \in \Gamma\}$. Recall that a graph G is acyclic if all its SCCs have only one element. In this case, the set of vertices of G can be partially ordered. We can now state Simon's result.

Theorem 2.15. *Let L be a language and $\mathscr{M} = (\Sigma, Q, i, F, \delta)$ its minimal automaton. L is piecewise testable if and only if the following two conditions hold*:
1. *The state transition graph G of $\mathscr{M}$ is acyclic.*
2. *For any subset Γ of Σ, each CC of $G_\Gamma = (Q, \delta_\Gamma)$ has a unique maximal state.*

3. Algorithms

In this section we will give a general outline of each of the algorithms implemented in the LANGAGE package, and supply a full description of our procedure for testing SLT languages. Maple affords sets, lists, and tables data types. Therefore the pseudo-code given for this last algorithm is very close to the Maple code.

For time complexity analysis, we denote by n the number of states of the automaton, by s the size of the alphabet and by m the number of edges of the graph of the automaton.

3.1. Algorithm for sLT languages

Owing to the notion of pair-graph introduced by Kim et al. [6] we state a lemma which yields an efficient implementation of Theorem 2.5.

Definition 3.1 (*Pair-graph*). Let $\mathcal{M}=(\Sigma,Q,i,F,\delta)$ be an automaton. Let Q_1 and Q_2 be two subsets of Q. Let $*$ be a symbol not in Q. The pair-graph on $Q_1\times Q_2$ is the edge-labeled directed graph $G(V,E)$, where $V=(Q_1\cup\{*\})\times(Q_2\cup\{*\})-\{(*,*)\}$ and E is defined as follows.

Define $\delta_i : Q_i\times\Sigma\to Q_i\cup\{*\},\ i=1,2$, such that for all $p\in Q_i$ and $a\in\Sigma$,

$$\delta_i(p,a)=\begin{cases} q & \text{if } \delta(p,a)=q\in Q_i,\\ * & \text{if } \delta(p,a)\notin Q_i.\end{cases}$$

Then $E=\{((p,q),a,(r,s)) \mid p\in Q_1, q\in Q_2, p\neq q, r\in Q_1\cup\{*\}, s\in Q_2\cup\{*\}, (r,s)\neq(*,*), \delta_1(p,a)=r, \text{ and } \delta_2(q,a)=s\}$.

Lemma 3.2. *Let C_i and C_j be two distinct SCCs of the state transition graph of an automaton $\mathcal{M}=(\Sigma,Q,i,F,\delta)$. Let Q_i and Q_j be respectively the set of states in C_i and C_j.*

1. *The component C_i is s-local if and only if the pair-graph on $Q_i\times Q_i$ has no cycle.*
2. *The components C_i and C_j are pairwise s-local iff the pair-graph on $Q_i\times Q_j$ has no cycle.*

sLT algorithm

(1) *Compute the SCCs of G.*
(2) *Let |SCC| be the number of SCCs.*
(3) **for** *i* **from**1 **to** |*SCC*| **do**
(4) **for** *j* **from** *i* **to** |*SCC*| **do**
(5) *Build the pair-graph G_{ij} on $Q_i\times Q_j$.*
(6) *Verify that $G_{i,j}$ has no cycle.* **lemma 3.2**
(* *otherwise exit, the language is not strictly locally testable* *)
(7) **endfor**
(8) **endfor**

Let us study the complexity of this algorithm. SCCs construction (1) can be done in $O(\max(m,n)) \leqslant O(n^2)$. Suppose that G has r SCCs and that n_j is the number of states of the SCC C_j. Computation of (4) and (5) is achieved in $O(sn_in_j)$. Thus we can assert that the complexity of this algorithm is $O(\sum_{i=1}^{r}\sum_{j=i}^{r} sn_in_j) \leqslant O(sn^2)$.

3.2. Algorithm for LT languages

We have implemented the algorithm due to Kim et al. [6] whose time complexity is $O(sn^2)$. The following lemma is particularily useful.

Lemma 3.3. *Let C_j be an s-local SCC of the state transition graph of a deterministic automaton $\mathcal{M}=(\Sigma,Q,i,F,\delta)$ and let C_0 be the reaching component of C_j. Let Q_0 and Q_j be the sets of states in C_0 and C_j and let G_{0j} be the pair-graph on $Q_0 \times Q_j$. Then the state transition graph of $\mathcal{M}$ is not TS-local w.r.t. C_j if and only if there is a path in G_{0j} from an SCC to a node of the form $(t,*)$ or $(*,t)$.*

Let G be the state transition graph of the automaton $\mathcal{M}$. Let C_0 be the component of the graph which consists of all the states from which C_j is reachable. Let Q_0 (resp. Q_j) be the set of states in C_0 (resp. C_j).

LT algorithm

(1) **Repeat**
(2) *Choose an SCC C_j of G without descendant.*
(3) *Compute C_0.*
(4) *Compute the pair graph G_{0j} on $Q_0 \times Q_j$.*
(* *make use of lemma 3.3 in order*
to perform the following test *)
(5) **if** *G is TS-local w.r.t. C_j* **then**
(6) $G := G - C_j$ (* *delete C_j* *)
(7) **else**
(8) *exit* (* *G is not locally testable* *)
(9) **endif**
(10) **until** *G has no SCC*

3.3. Algorithm for SLT languages

First we will introduce two definitions and a new theorem from which we deduce our algorithm for testing whether or not an automaton recognizes a strongly locally testable language.

Definition 3.4 (*Product-graph of an SCC*). Let $\mathcal{M}=(\Sigma,Q,i,F,\delta)$ be an automaton. Let $C \subseteq Q$ be an SCC of $\mathcal{M}$. The product-graph of the SCC C is the directed graph $G(V,E)$ where $V=\{(p,q)=(\delta(i,w),\delta(r,w)) \mid r \in C,\ \ \delta(r,w) \in C,\ \ w \in \Sigma^*\}$ and $E=\{((p,q),a,(p',q')) \mid a \in \Sigma,\ \ \delta(p,a)=p',\ \ \delta(q,a)=q'\}$.

Definition 3.5 (*Attracting point*). Let $G=(X,U)$ be a directed graph. An attracting point is an SCC which has no descendant.

Theorem 3.6. *Let L be a locally testable language and let $\mathscr{M}=(\Sigma,Q,i,F,\delta)$ be its minimal complete automaton. The following two properties are equivalent:*

1. *For every SCC C of $\mathscr{M}$, the product-graph of C has exactly one attracting point.*
2. *For every SCC C of $\mathscr{M}$, we have $L_C \subseteq P_C$ or $L_C \cap P_C = \emptyset$.*

Proof. We first prove $1 \Rightarrow 2$ by showing that if $L_C \not\subseteq P_C$ and $L_C \cap P_C \neq \emptyset$ then we have a word $w \in L_C \cap P_C$. This word is the label of two paths leading to the state $p \in C$ (one from the initial state, another from a state of the SCC C). It comes that (p,p) is a state of an attracting point of the pair-graph. In a second time, we prove that for a word $w' \in L_C$ and $w' \notin P_C$, we can build a state (p_1,q_1) in the pair-graph. There is no path from (p_1,q_1) to (p,p) so there is a second attracting point. The proof of the converse is in the full paper. □

SLT algorithm is directly deduced from this theorem. The function *product-graph* computes the product-graph of a SCC C. It is based on a universal generation technique. Starting from each of the vertices (i,e) where e is a state in C, we produce the vertices of the product-graph according to the Definition 3.4.

```
function product-graph(C, M)
X ← ∅
foreach e in C do
        X ← X ∪ {(1,e)}
endfor
Xt ← X
while Xt ≠ ∅ do
            take (e,e') in Xt
            foreach letter in Σ do
                if δ(e',letter) ∈ C then
                        f ← δ(e,letter)
                        f' ← δ(e',letter)
                        if (f,f') ∉ X then
                                X ← X ∪ {(f,f')}
                                Xt ← Xt ∪ {(f,f')}
                        endif
                        U ← U ∪ {((e,e'),(f,f'))}
                endif
            endfor
endwhile
return (G=(X,U))
end
```

The function *one-attracting-point* lies on a transitive closure computing. If there exists a unique attracting point C, any state of C can be reached from any state of G. So we compute the transitive closure of each state of G and verify that the intersection is not empty.

```
function one-attracting-point(G = (X, U))
Inter ← X
foreach x in X do
        transitive[x] ← ∅
endfor
foreach (x, y) in U do
        transitive[x] ← transitive[x] ∪ {y}
endfor
foreach x such that transitive[x] ≠ ∅ do
        K ← transitive[x]
        while K ≠ ∅ do
                Tm ← transitive[x]
                foreach y in K such that (y,z) ∈ U do
                    transitive[x] ← transitive[x] ∪ {z}
                endfor
                K ← transitive[x]\Tm
        endwhile
        Inter ← Inter∩transitive[x]
endfor
return(Inter ≠ ∅)
```

We can now state the algorithm for testing whether or not an automaton recognizes a strongly locally testable language.

SLT algorithm

```
(1) Check if 𝒜 is locally testable
(2) Compute the SCCs C_i of 𝒜
(3) for i from 1 to |SCC| do
(4)       G ←product_graph(𝒜, C_i)
(5)       if one-attracting-point(G) = false then
(6)           return(false)
(7)       endif
(8) endfor
(9) return(true)
```

The complexity of local testability test is in $O(sn^2)$. We test the conditions (4) and (5) in $O(\sum_{i=1}^{r} s|Q| \times |Q_i|) \leqslant O(sn^2)$. Hence the complexity of SLT algorithm is $O(sn^2)$.

3.4. Algorithm for PT languages

This algorithm is described by Stern in [11] and has a $O(sn^5)$ time complexity. To each state q is associated the set

$$\Sigma(q) = \{a \in \Sigma \mid \delta(q,a) = q\}.$$

If q and q' are distinct maximal states of C then they are also distinct maximal states of some component of $G = (Q, \delta_\Gamma)$ where $\Gamma = \Sigma(q) \cap \Sigma(q')$, hence condition (2) of Theorem 2.15 can be restricted to the subsets Γ of the form $\Sigma(q) \cap \Sigma(q')$.

Proposition 3.7. *Let G be the state transition graph of a minimal deterministic automaton $\mathcal{M} = (\Sigma, Q, i, F, \delta)$. If G is acyclic then $q \in Q$ is a maximal state of a component C of $G_\Gamma = (Q, \delta_\Gamma)$ iff*

1. $q \in C$.
2. $\Gamma \subseteq \Sigma(q)$.

PT algorithm

```
(1)  Find all SCCs of M.
(2)  for each SCC C_i do
(3)       verify that C_i is acyclic
(4)  endfor
(5)  for q in Q do
(6)       compute Σ(q)
(7)  endfor
(8)  for q in Q do
(9)       for q in Q do
(10)           if q ≠ q' then
(11)               compute G' = (Q, δ_Γ) where Γ = Σ(q) ∩ Σ(q')
(12)           if Γ ≠ ∅ then
(13)               compute G'' = (Q, U) the transitive closure of G'
(14)           for p in Q do
(15)               if (p,q) ∈ U and (p,q') ∈ U then
(16)                   exit (* L is not piecewise testable *)
(17)               endif
(18)           endfor
(19)       endif
(20)   endif
(21)   endfor
(22) endfor (* L is piecewise testable *)
```

4. Conclusion

LANGAGE is the third package of AGL, the new version of AG [5]. An algorithm of conversion from a "Glushkov" automaton to the language it recognizes is also implemented in LANGAGE. The whole package is to be interfaced on the World Wide Web. In a first time, this interface will allow users to input regular expressions. Next, graphical inputs will be possible. Some other tests of languages are already investigated. It is the case of the threshold locally testable languages which are introduced in [2]. This package as well as AG is available via anonymous ftp at *ftp.dir.univ-rouen.fr* in the directory *pub/MAPLE/AG*.

References

[1] D. Beauquier, J. Berstel, P. Chrétienne, Éléments d'Algorithmique, Masson, Paris, 1992.
[2] D. Beauquier, J.-E. Pin, Scanners and languages, Theoret. Comput. Sci. 84 (1991) 3–21.
[3] J.A. Brzozowski, I. Simon, Characterization of locally testable events, Discrete Math. 4 (1973) 243–271.
[4] P. Caron, Families of locally testable languages, Theoret. Comput. Sci., to appear.
[5] P. Caron, AG: A set of Maple packages for manipulating automata and finite semigroups, Software – Practice & Experience 27(8) (1997) 863–884.
[6] S.M. Kim, R. McNaughton, R. McCloskey, A polynomial time algorithm for the local testability problem of deterministic finite automata, IEEE Trans. Comput. 40(10) (1991) 1087–1093.
[7] R. McNaughton, Algebraic decision procedures for local testability, Math. Systems Theory 8 (1974) 60–76.
[8] R. McNaughton, S. Papert, Counter Free Automata, MIT Press, Cambridge, MA, 1971.
[9] M.-P. Schützenberger, On finite monoids having only trivial subgroups, Inform. Control 8 (1965) 190–194.
[10] I. Simon, Piecewise testable events, in: Proc. 2nd GI Conf., Lecture Notes in Computer Science, vol. 33, Springer, Berlin, 1975, pp. 214–222.
[11] J. Stern, Complexity of some problems from the theory of automata, Inform. Control 66 (1985) 163–176.
[12] Y. Zalcstein, Locally testable languages, J. Comput. System Sci. 6 (1972) 151–167.